Concepts, Design and Implementation of the ATLAS New Tracking (NEWT)

T. Cornelissen, M. Elsing, I. Gavrilenko
CERN
S. Fleischmann
University of Bonn, Germany
W. Liebig
NIKHEF, Amsterdam, The Netherlands
E. Moyse
University of Massachusetts, USA
A. Salzburger (Editor)*
Leopold Franzens Universität Innsbruck, Austria & CERN

for the ATLAS Inner Detector Software Group

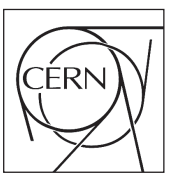
December 17, 2007

ATL-SOFT-PUB-2007-007
17 December 2007

Abstract

The track reconstruction of modern high energy physics experiments is a very complex task that puts stringent requirements onto the software realisation. The ATLAS track reconstruction software has been in the past dominated by a collection of individual packages, each of which incorporating a different intrinsic event data model, different data flow sequences and calibration data. Invoked by the *Final Report of the Reconstruction Task Force*, the ATLAS track reconstruction has undergone a major design revolution to ensure maintainability during the long lifetime of the ATLAS experiment and the flexibility needed for the startup phase. The entire software chain has been reorganised in modular components and a common *Event Data Model* has been deployed during the last four years. A complete new track reconstruction that concentrates on common tools aimed to be used by both ATLAS tracking devices, the Inner Detector and the Muon System, has been established. It has been already used during many large scale tests with data from Monte Carlo simulation and from detector commissioning projects such as the combined test beam 2004 and cosmic ray events. This document concentrates on the technical and conceptual details of the newly developed track reconstruction, also referred to as *New Tracking*.

ATLAS NOTE
The ATLAS Experiment, *http://www.atlas.ch*



*corresponding author: Andreas.Salzburger@cern.ch

1 Introduction

The event reconstruction of modern high energy physics experiments is a very complex task but indispensable for any analysis of the underlying physics process. At the LHC with its design luminosity of 10^{34} $\mathrm{cm}^{-2}\mathrm{s}^{-1}$ and multiple overlaying proton-proton collisions the resulting high track density puts stringent challenges to the track reconstruction software. The ATLAS detector consists of two independent tracking devices: the *Inner Detector* (ID) close to the interaction region — realised as the pixel detector at innermost radii, *Semiconductor Tracker* (SCT) using silicon strips, and a *Transition Radiation Tracker* (TRT) — and the *Muon System* (MS), that combines *Monitored Drift Tubes* (MDT), *Cathode Strip Chambers* (CSC), *Thin Gap Chambers* (TGC), and *Resistive Plate Chambers* (RPC). While the Inner Detector reconstruction has to deal with the high track density that imposes a large number of combinatorial track candidates, the Muon System track reconstruction is mainly limited by the huge amount of inert material, the cavern background and the highly inhomogeneous magnetic field.

In the past[1], the ATLAS track reconstruction software consisted for both, ID and MS, of several competing reconstruction programs, each of which incorporating its own data model, different reconstruction geometries, varying concepts in material integration and separate philosophies in algorithmic sequence and steering[2]. Performance comparisons between the various programs on different levels of the reconstruction chain have been almost impossible. New developments in terms of calibration, conditions or alignment data that are needed for a more realistic description of the ATLAS detector had to be integrated separately into the existing software packages, since the use of common tools was not incorporated into the underlying software design.

The long lifetime of the ATLAS experiment, however, requires a reconstruction software that is flexible, extendable, easy to maintain or change, whilst keeping the performance at the highest and the CPU time consumption at the lowest possible level. The *ATLAS Reconstruction Task Force* (RTF) recognised in 2003 [1] the potential danger of sticking with the monolithic, and sometimes even single-author software structure, and developed a guideline for the transition of the ATLAS reconstruction software to a modern, modular pattern that is based on common interfaces and a shared *Event Data Model* (EDM). During the last four years this shared EDM has been developed and established [2], which builds a cornerstone of the ATLAS *New Tracking* (NEWT). Given the modular software structure, *New Tracking* is not a name for yet another reconstruction program, but rather a collective term for the philosophy that has been deployed throughout the new track reconstruction software. A lot of the existing and well performing code that has been part of the past reconstruction programs has been integrated as components into the new modular design. Currently, NEWT spans over about 250 software packages, located in the common Tracking repository as well as in the associated sub-detector repositories, and concentrates the work of many developers. NEWT has been developed in full coherence with the ATLAS software framework ATHENA and respects ATLAS coding standards [3]. This document is based on the ATLAS software release 12.0.6. Developments that have been integrated after this release are clearly marked within the context.

1.1 Document Structure

This document is organised as follows: Section 2 gives an introduction to the ATHENA framework, concentrating on the concepts and modules used in the realisation of New Tracking, Sec. 3 focuses on the high level structure of the ATLAS New Tracking in both algorithm sequence and data flow; additionally, an overview of the most common Tracking interfaces and tools is given. Section 4 describes the track reconstruction strategies and their implementation in the current ID New Tracking. Section 5 describes the usage of the common tools in the Muon System and in the context of muon combined reconstruction. Finally, Sec. 6 will give a conclusion, but will also present an outlook of the ongoing evolution of the New Tracking realm. The appendix explains typesetting and gives a brief overview on the used software packages such as an index of used abbreviations within this document for the orientation of the reader.

[1] Starting from the design phase of the ATLAS experiment.

[2] Commonality has not even been met in the used programming language, as for historical reasons these track reconstruction programs have been written in `C`, `C++`, `Fortran` or even a mixture of all three.

2 The embedding Software Framework ATHENA

The main concept of NEWT is to factorise the complex process of track reconstruction into well defined modules that represent a single task or a grouped operation. A common interface structure for these modules has been defined and the EDM acts hereby as the language between the different components. This allows single parts of the entire reconstruction process to be modified or exchanged without disrupting the untouched parts of the software chain.

To understand the structure of the ATLAS New Tracking, it is necessary to be familiar with the main concepts of the underlying software framework, not necessarily on a detailed technical, but on an abstract level. The following section should give the reader enough knowledge to understand the concepts of algorithmic sequences and the overall data flow, while attempting to minimise the purely technical aspects of the ATLAS software structure. The reader is, however, encouraged to read further in [4], Sec. 3.3, where a complete overview of the ATHENA framework is given.

2.1 The ATHENA Framework and the Component Architecture

The ATLAS ATHENA software framework is an enhanced version of the original C++ based software framework GAUDI [5] that was initially developed by the LHCb collaboration. Nowadays, the two projects have been merged to a joint ATLAS-LHCb project, sometimes referred to as GAUDI-ATHENA.

2.1.1 The ATHENA Component Pattern Architecture

The component model is a very common software architecture of large-scale software projects. Commonly used interfaces are defined and allow various alternative realisations — i.e. in software terms, different implementations — of the same task. The component library structure allows that these alternative modules are loaded as shared libraries at job configuration level. This leads to a flexible software structure and reduces in addition dependencies between the various libraries used in the final executable, which in turn increases the stability and decreases the complexity of the build process of the ATLAS software[3]. Further information about the ATLAS software model can be found in [6].

The ATHENA framework provides a set of defined interfaces at different levels in the program sequence that also build the base pattern of the ATLAS New Tracking:

- The `Service` class is designed to provide dedicated functionality during the entire program execution, e.g. the central data storages (transient event store or the detector store) are realised as ATHENA `Service` objects. `Service` instances are handled by a central `ExtSvc` manager, that regulates initialisation and finalisation.
- The `Algorithm` class is dedicated for actions to be taken exactly one time for every processed event, e.g. most of the data preparation algorithms are realised as `Algorithm` classes. These have to be registered to a central `ApplicationMgr` in the job configuration that steers initialisation, finalisation and the execution of the `Algorithm` at every event.
- Unlike the `Algorithm` class provides the `AlgTool` are more flexible solution for repeated operations, mainly called through an `Algorithm` that either owns the associated `AlgTool` (i.e. consequently a *private* `AlgTool`) or retrieves it from the central `ToolSvc`, where all *public* tools are registered. This pattern allows `AlgTool` instances to be shared between different applications, such as e.g. the same `Extrapolator AlgTool` instance is used at several places within the reconstruction process.

In general, the `Algorithm` is responsible for retrieving input data collections from and writing the output data to the transient event store, which is realised through the ATHENA `Service StoreGateSvc`. The actual operations are usually not performed by the `Algorithm`, but delegated to `AlgTool` units, that can be shared between different tasks. Per event, the various `Algorithm` instances are called one

[3] In the ATLAS computing model, test builds of the entire software suite take place on a daily basis to allow a coherent and parallel code development of the multiple authors. Additionally, it increases the frequency of the development cycles and allows large scale evolutions of the software in a gradual way.

after the other by the `ApplicationMgr` through which the reconstruction sequence is defined. There is no additional dependency between the different `Algorithm` classes rather than they rely on the input data to be existing and retrievable through the `StoreGateSvc`. Following [7], this data centered architecture will be further on referred to as *blackboard architecture style*, where the `StoreGateSvc` acts as the blackboard to which the clients write to or read from (in ATHENA, this is represented by the templated `StoreGateSvc::retrieve()`, respectively `StoreGateSvc::record()` interface). The `ApplicationMgr` plays the role of a *controller* (teacher) in this model and organises the reading and writing to and from the blackboard. The result of the blackboard design can be seen as a *pseudo data flow*: in an abstract picture the data objects are handed over from one `Algorithm` to the next one in the reconstruction sequence, while in reality the data exchange always progresses via the blackboard.

Figure 1 shows a schematic UML sequence diagram for a sample usage of the three main framework components in a simple processing example.

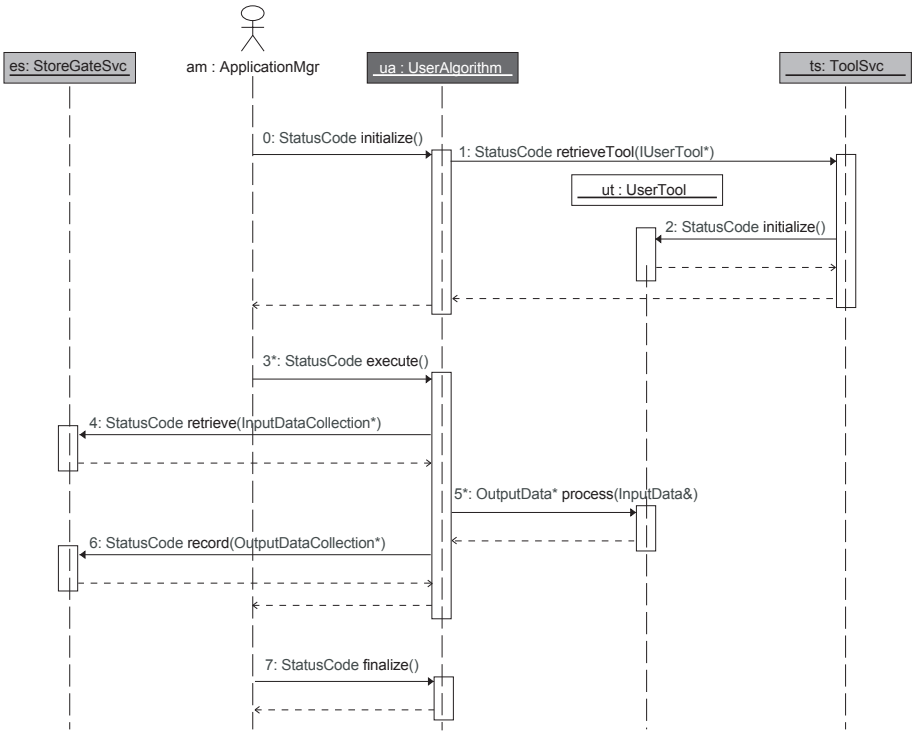


Figure 1: A simplified UML sequence diagram for the data flow and sequence steering in the ATHENA framework. Every `Algorithm` is registered to the `ApplicationMgr` that calls the `initialize()` interface method at job startup, the `execute()` for every event and the `finalize()` method in the job termination phase. The `initialize()` call to the `Algorithm` may trigger the retrieval of an `AlgTool` from the `ToolSvc`. The main framework methods are templated and return a `StatusCode` object that indicates the success state of the performed operation.

2.1.2 Data Factory Pattern and Encapsulation of algorithmic Code

A main concept on the ATLAS computing model — and a natural consequence of the blackboard design — is a clear and strict separation between data classes and algorithmic code, that is mainly carried out by `AlgTool` classes. Operations are performed on input data classes, and newly created output data is hereby produced. In general, modifications to the input data are not allowed, but the modification of an object leads rather to the creation of a new one. In C++ terms, these new objects are dynamically created using the `new` operator. This concept is intrinsically imposed by the blackboard framework design that is concentrated around a central data store service: allowing modifications of an object would change the object immediately at every appearance in the reconstruction chain and since there is no relation between the different `Algorithm` modules, a direct manipulation of EDM objects could affect any decoupled `Algorithm` that uses the same input sample without being

triggered. This is the reason why the tracking EDM classes in general do not have modification or set methods which are commonly used in many other application libraries[4]. This model will in the following be referred to as *data factory pattern*.

3 Concepts and Common Components of the New Tracking

In a complex software project of large scale that serves both users and developers on different levels of interaction, it is inevitable that common structures and patterns are identified and named, such that an intuitive guideline through the — for the single user — almost unmanageable matrix of components is present[5]. Section 3.1 tries in the following to define these common structures in an abstract way, while Sec. 3.2 concentrates on the actual used common components in the context of the ATLAS New Tracking reconstruction.

3.1 Reconstruction Sequences, Modules, Tasks and Operations

In the following, the components of the track reconstruction software will be classified as sequences, modules, tasks and operations:

- A *sequence* specifies a complete process of several complex reconstruction steps leading to high level output objects; track finding with two strategies may be performed through two different sequences, established as different `Algorithm` chains.

- A *module* denotes, in general, the complete chain of retrieving input data from the transient event store, processing the data, and the successive recording of the output data. Most data preparation processes are realised as modules; another example would be track refitting, where a complete track collection is retrieved from the `StoreGateSvc` and a new collection of tracks is recorded to it after refitting with a track fitter. Modules are in general realised as `Algorithm` classes with their associated access to the transient event store.

- A *task* describes on the other hand a well-defined execution without direct interaction with the algorithm sequence. The refit of a track, e.g. is a task in this scope and a module consists usually of one to several tasks. Tasks are often performed by `AlgTool` classes which are directly called by an `Algorithm`; such `AlgTool` instances will be in the following called *at first level*.

- The lowest component to be mentioned in this context is an *operation*, usually performed by an `AlgTool` object of lower level. Operations are often shared between tasks (and a task is in general built of several operations) and performed several times, but do — contrary to a task — not lead to a direct output data object. The extrapolation of a track representation to surfaces, which is needed in track fitting as well as in pattern recognition, vertex fitting and combined reconstruction is an example for such an operation.

Lower level algorithmic code or utilities are not forced into the `AlgTool` schema since this creates a quite substantial overhead to a simple class definition: small mathematical utility classes, functors for object sorting and access, but also little manipulator `friend` classes are therefore concentrated in dedicated utility packages that do not embody the component library pattern, but are of an installed library type[6].

[4] For low level objects that have not been written to the transient event store or those that are scheduled for immediate further processing it is, however, sometimes useful when small modifications are allowed without necessarily creating a new object and destroying the old one. In this case, dedicated helper classes that are in a C++ `friend` relation to the data objects can be called that perform the manipulation, while respecting the self-consistency of the object.

[5] The ATLAS offline release has already exceeded the number of 1000 component packages.

[6] The major difference between *component* libraries and *installed* libraries is their role in the linking process of the software build. While component libraries can be loaded dynamically at run time as shared libraries, installed libraries have to be linked against. In more illustrative words: when one deals with goods it is necessary to know their size and shapes, while it is not obligatory to about the dealer who delivers them.

3.2 Common Interfaces and Tracking `AlgTool` Classes

The framework `Algorithm` class builds the natural interface for a module, while operations and tasks impose a higher granularity and have to be defined by an interface structure that allows multiple actual implementations. In the ATLAS New Tracking, there exist two different types of interface definitions for `AlgTool` objects: *common interfaces* that define operations and tasks performed on base class level of the tracking EDM and *specific interfaces* that describe operations that are particular to a sub-detector. Common interface definitions are concentrated in the Tracking repository, while specific interfaces are spread out over the sub-detector repositories. Common interfaces do not necessarily imply that the implementations can be kept in an generic way or an abstract level. Moreover, some operations and tasks will be best performing if a very specific and for the sub-detector technology optimised way can be found. In this case, the actual `AlgTool` classes are located in the sub-detector repositories and the method interface is, if possible, overridden such that the base class EDM object is hidden by the concrete sub-detector implementation. Table 1 gives an overview of the locations of common interfaces defined in the Tracking repository.

Table 1: Locations and brief description of common interfaces concentrated in the Tracking repository. For future releases it is scheduled to move the track fitting interfaces into a dedicated TrkFitterInterfaces package to sustain a coherent naming schema.

Interface package	Description	Example
TrkExInterfaces	propagation, extrapolation, material effects	`IPropagator`
TrkDetDescrInterfaces	building of the `TrackingGeometry`	`IGeometryBuilder`
TrkMagFieldInterfaces	magnetic field access, parameterisations	`IMagneticFieldTool`
TrkFitterUtils	fitting interfaces and extensions	`ITrackFitter`
TrkToolInterfaces	truth processing, updator, ...	`IUpdator`
TrkValInterfaces	validation tools	`IResidualPullCalculator`
TrkVertexFitterInterfaces	vertex seed finding, fitting	`IVertexFitter`

The list of specific interfaces that define tasks and operations in the sub-detectors is wide-spread, every single `AlgTool` is represented through a dedicated interface even if only one single implementation is currently present[7]. This allows code development beyond the level of pure structured programming, but it also necessary for the component pattern design. Modularity is hereby achieved through the fact that all interactions between modules are kept purely on interface level.

3.3 The ATLAS Tracking Event Data Model and Reconstruction Geometry

A common EDM is inevitable for a modular software architecture. It allows the definition of abstract interfaces by identifying similar tasks to be performed through the method signature and the return type of the method, respectively. Different concrete implementations of the same abstract interface class may be created for e.g. different sub-detectors, realising a similar operation with different but optimised implementations.

During the last four years a common ATLAS tracking EDM has been deployed, concentrated around a very flexible and extensible `Track` class. The collection of `Track` objects is capable of holding the entire tracking information of the event. The polymorphic inheritance structure of the ATLAS tracking EDM enables the definition of tasks that can be performed on base class level, or — if detailed information about the actual data type is required — on objects of concrete type. Even a brief description of the EDM classes that are used in NEWT would go far beyond the scope of this document, but is essential for the understanding of the implementation of the New Tracking chain. The reader is therefore encouraged to find all detailed information about the EDM in [2]. The main tracking EDM classes, such as the track and measurement representations will in the following be used without further description of their class structure.

[7] The definition of tasks and operations through interfaces allows to evaluate, modify and eventually exchange the single modules of the track reconstruction during the long term operation of the ATLAS detector.

Together with the new EDM a common reconstruction geometry has been developed coherently to guarantee a consistent geometrical description for the use in track reconstruction. A core `Surface` class description has been introduced that builds the interface of geometrical information (given by the central ATLAS detector description) with tracking relevant data, such as hits on detecting surfaces. In other words, the `Surface` builds the binding link between the geometry and the linear algebra applications of the EDM. The purely mathematical description given by the `Surface` classes is extended to `Layer` and `TrackingVolume` object that give access to the material and magnetic field information. The full reconstruction geometry will be in the following referred to as `TrackingGeometry` and is described in detail in [8].

Common Tools Many repeating tasks and operations during track reconstruction can be performed without the need of any dedicated information about the underlying detector technology. This is in particular given for purely mathematical operations or higher level tasks that operate on base class level of the underlying tracking EDM. Track and vertex fitting are, in general, independent from the way the track information has been gathered as long as the EDM provides enough — and mathematical consistent — information about the track to perform the fit. In the ATLAS New Tracking realm, these tasks have been identified and concentrated in common `AlgTool` implementations, ordered in an intuitive package structure in the Tracking repository. The main tools and concepts are described in the following sub-sections. The building of the reconstruction geometry, which is also realised in the common interface structure of the New Tracking reconstruction is omitted in this consideration, since it is done only once at startup of the reconstruction job.

3.3.1 The Extrapolation Engine

The transport of track parameters (i.e. the representation of a track with respect to given detector surfaces) is a very frequent process in track reconstruction. It can be performed with different complexity, following diverse track models and propagation techniques. In many tasks, special consideration has to be paid on the correct integration of the effects originating from interactions of the particle with the traversed detector material, while in others this is of minor importance. The ATLAS extrapolation package provides a very flexible set of `AlgTool` implementations, including propagators for the purely mathematical transport of the track parameters, classes for material effects integration and magnetic field access. The extrapolation package is located in the TrkExtrapolation CVS repository and is based on the newly developed reconstruction geometry package. Further details about the extrapolation package can be found in [9].

3.3.2 Magnetic Field Access

The access to the magnetic field information in ATHENA is done by a dedicated `Service`, the `MagFieldAthenaSvc`. In the pattern finding stage of track reconstruction it is usually sufficient to use a less granular parameterisation of the magnetic field, while in the final track fit, the best description of the magnetic field is used to achieve the optimal track resolution. In NEWT, a dedicated `AlgTool` interface layer between the standard ATHENA magnetic field access service and the client code has been inserted to allow specific simplifications, modifications of the given magnetic field map or the direct access to the ATHENA `MagFieldSvc` through one single interface. For the extrapolation package the access to the magnetic field tool is given through a special `MagneticFieldProperties` class. The `MagneticFieldProperties` class is a base class of the `TrackingVolume` class, which allows the definition of different field configurations for different parts of the detector. This is of special interest for many pattern recognition programs where simplified parameterisations provide, in general, a satisfactory accuracy for the pattern search. The additional flexibility of modifications and distortions will be in particular important for the scaling and adjustment of the magnetic field during the start-up phase of the ATLAS experiment and did proof well during data taking of the ATLAS combined test beam in 2004.

3.3.3 The `ITrackFitter` Interface and common Track Fitters

A track fit is the estimation of the best set of parameters describing the track with respect to a given reference surface when a collection of hits is given to define the track trajectory. This fitting procedure can be performed in various different ways, while the input data to the fit is properly defined: fitting can be done on a set of non-calibrated hits (i.e., in the ATLAS EDM, a collection of `PrepRawData`) objects, on a set of calibrated measurements of various types (in ATLAS realised as extensions of a `MeasurementBase` base class) or as a refit of a given `Track` or `TrackSegment` object. Only few additional parameters, such as a particle type hypothesis for the material effects integration or a possible outlier logic steering are needed to specify the fitting procedure. In NEWT, a general `ITrackFitter` interface has been imposed that defines the main functionality of any used track fitter; currently six different fitting techniques are implemented under this interface and can be chosen at job configuration level:

- `KalmanFitter` (KF): the `KalmanFitter` is a straight-forward implementation of the Kalman filter technique that has been adopted for the track fitting in high energy experiments [10]. It combines forward filtering, backward smoothing and an outlier rejection; it uses the extrapolation engine with its underlying reconstruction geometry for filter step predictions. For the ATLAS silicon detector, the KF has a dedicated extension for he fitting of electron tracks, that lose stochastically a significant part of their energy due to bremsstrahlung effects. In this case, the purely Gaussian process noise assumption — for energy loss based on ionisation loss — that is intrinsic to the best estimator mechanism of the Kalman approach is far from being optimal. A special dynamic noise adjustment schema (DNA) [11] has been developed that still uses a Gaussian error assumption, but adapts the value of the applied variance with respect to the amount of traversed material.

  The gain matrix driven update of a track prediction with a given measurement is the main concept of the Kalman filter technique. In NEWT, this specific operation is defined by an `IUpdator` interface and accessed by the `KalmanFilter`. The different `IUpdator` realisations that exist in the ATLAS New Tracking realm are described in Sec. 3.3.4.

- `DeterministicAnnealingFilter` (DAF): the deterministic annealing technique [12] combines the standard Kalman filter formalism with a probabilistic description of the measurement assignment to a track; per detecting surfaces several measurements can be fitted at once, each weighted by the current assignment probability given through the track fit. An annealing schema using a defined Boltzman function is performed by *cooling* the system *temperature* to a threshold level while iterating the track fit. This schema allows the DAF to perform local pattern recognition in combination with a track fit in detector regions with high hit multiplicities. The DAF is implemented using the `KalmanFitter` underneath and imposing the additional annealing schema. The DAF extends the tracking EDM with a probabilistic hit description, i.e. multiple hits are grouped together in one single measurement class. Details about the actual implementation of the DAF concept in the ATLAS New Tracking realm can be found in [13].

- `GaussianSumFilter` (GSF): the GSF is a special multi-Gaussian extension of the standard Kalman fitter [14], aimed at the reconstruction of electron tracks. In the GSF approach, the highly non-gaussian probability density function of the electron energy loss is modeled by a mixture of several Gaussians. To integrate this model into the track parameterisation, the track parameters for the GSF are realised as a multi-component bundle and processed simultaneously. Evidently, the GSF needs an extended EDM for handling the multi-component approach. Component reduction is imposed at several steps to avoid an exponential growth of the number of components describing the track during the full track fit.

- `AlignmentKalmanFitter` (AKF): the alignment of the ATLAS detector will be a challenging task that is of particular interest at the startup phase of the experiment. Track based alignment procedures play hereby an important role and will be carried out trough the entire lifetime of the experiment. Recently, an extended version of the Kalman filter has been developed [15] that integrates the update of the detector surface orientation and position into the intrinsic measurement update of a Kalman filter step. This technique is realised in ATLAS through a special `AlignmentKalmanFitter` that is based upon the standard `KalmanFitter` implementation

and an extended version of the ATLAS reconstruction geometry that introduces custom alignable `Surface` objects[8].

- `GlobalChi2Fitter`: the track fit through the minimisation of a global χ^2 value is a very common and robust fitting technique that is described in various publications (such as [16]). Given purely Gaussian process noise, the minimisation of the χ^2 value that is built from the hit residuals at every measurement surface marks the best set of estimators of the track trajectory. Material effects enter the χ^2 function as additional fitting parameters, weighted by their expected variance due to their stochastic behavior. The minimisation of the global χ^2 value is then, in general, performed by solving a set of linear equations through a matrix inversion. Thus, the number of global parameters have to be kept low to minimise the CPU time consumption. In NEWT, this technique is implemented through the `GlobalChi2Fitter`, which has been of extensive use while reconstructing data from the combined test beam 2004 and during the reconstruction of tracks originating from cosmic rays, see Sec. 4.4. The `GlobalChi2Fitter` is interfaced with the common reconstruction geometry via a dedicated dynamic layer schema.

- `DistributedKalmanFilter` (DKF): this is a modified version of the the Kalman filter formalism that estimates the track parameters only for the perigee representation. This leads to a significant speed-up of the algorithm, since the for the original Kalman approach necessary propagations of the state vector to the measurement surfaces are reduced to a pure integration of material effects. The DKF also deploys a χ^2 based outlier rejection and an internal node schema for representing barrel or endcap measurements. It has been designed mainly for the use in the ID LVL2 Trigger and Event Filter and is a substantial part of the ID LVL2 application IDSCAN [17].

3.3.4 The `IUpdator` Interface

The update of the predicted track parameters with a measurement is a commonality of most progressive track fitting algorithms. Different models based on a covariance or weight matrix formalism can hereby be used. The mathematical background for the measurement update is well defined, however, the realisations can differ through approximations. In NEWT a dedicated `IUpdator` interface is provided and four different `AlgTool` implementations exist: three of which incorporate a covariance matrix based formalism and differ mainly through the used underlying math library and numerical stability. The fourth `IUpdator` implementation deploys a weight matrix approach and is superior in timing performance when being used with the DAF since it saves unnecessary matrix inversion when weighting the individual measurements.

The `IUpdator` can also be used to perform a reverse measurement update, as long as the full information on the measurement surface (including the combined covariance matrix) is given. This allows to calculate unbiased hit residuals without performing an additional track fit, which is a useful feature for the validation and alignment algorithms.

3.3.5 Calibration on Track

One concept of the ATLAS New Tracking is the (re-)calibration of the measurement based on the track direction and sensor intersection point. This can be used e.g. for the definition of the drift radius sign, error scaling, future calibration or conditions data depending on a given module intersection such as dead or noisy channel information. The imposed calibration model includes the adaption of cluster errors as well as the integration of chamber or module distortions to account for a realistic geometry description. Every sub-detector technology in ATLAS will have different strategies for calibration, optimised for performance of the individual part. To integrate the *on track* calibration into the general track fit, it is necessary to provide a schema that is at highest level independent from the used concepts and encapsulated from the sub-detector realms. This is established through a mixture of a common interface and various different implementations in the sub-detector repositories.

[8] The `Surface` objects of the ATLAS reconstruction geometry are fully integrated into the conditions data schema of ATLAS and retrieve alignment data at geometry construction or triggered through a callback in case that the geometry setup has changed during a single reconstruction job. The `AlignableSurface` that extends the common `Surface` base class introduces, however, a user-open alignment update possibility which is needed for this iterative alignment approach.

The IRIO_OnTrackCreator defines this transition from non-calibrated (PrepRawData) to calibrated (RIO_OnTrack) EDM objects and finds a common implementation in the Tracking realm: the so-called RIO_OnTrackCreator, holds a collection of pointers to further IRIO_OnTrackCreatorTool objects, each of which representing a different sub-detector.

3.3.6 Summary, Scoring and Helper Tools

The ATLAS tracking EDM provides various models of classifications for tracks: the most widely used quality information of fitted tracks is the χ^2 value together with the number of degrees of freedom n_{dof} of the track fit. This information is directly accessible through the Track object, but does not contain a lot of information about the track morphology. The χ^2/n_{dof} is a good parameter for a fast separation of good and bad tracks or for the application of quality cuts on a given track sample. However, it does not help in classifying the tracks any further which is necessary at various stages in the track reconstruction (i.e. the identification of fake tracks, the solving of hit ambiguities or evaluation of track extensions). Detailed information about the track characteristics in the sub-detector has to be accessible, which in the ATLAS EDM is realised through a TrackSummary object. The New Tracking uses a dedicated track scoring approach: first the TrackSummary is created by the TrackSummaryTool, which parses the Track object and records the hit statistics and characteristics for the various sub-detectors into a pre-defined enumeration schema. Helper tools in both, ID and MS, guarantee hereby the access to the relevant information that is specific to the detector technology. The TrackSummary is not assigned to a Track object as a private member, since many of the hit characteristics, such as shared hits, are a property of the processed track collection and not of a single track. Dedicated scoring functions that give bonus or penalty points for different patterns calculate a final track score that is then used for the track classification. Since the ITrackScoringTool allows different implementations of scoring functions that may also incorporate a different scoring system, the track score is not stored on track to prevent comparisons of track scores from different sources and with different meanings.

3.3.7 Truth Association and Validation

The validation of the entire track reconstruction chain is a necessary but complex task, since many different processes contribute to the final track reconstruction result. In the ATLAS New Tracking realm, dedicated emphasis has been put on having automated validation procedures for different stages of the track reconstruction process. The common EDM allows to concentrate the validation algorithms into a separate structure, while supporting the input of several different track reconstruction sequences as long as they comply with the tracking EDM.

Truth Association For many validation studies using Monte Carlo simulated data the association of reconstructed input data with the Monte Carlo truth is essential. The ATLAS tracking EDM follows a strict association pattern for the truth binding, i.e. no direct link between the event data object and the corresponding Monte Carlo truth object exists, but the relation between the two is purely done by an associative container. Several sets of Algorithm and AlgTool classes exist that parse given EDM input containers and establish the association to the truth objects. The truth association is mainly done on three different levels:

- **Hit Truth Association:** the clusters and drift circles are in ATLAS represented by PrepRawData objects. The first stage in the truth association of the track reconstruction is thus to find the relation of the PrepRawData objects to the simulated hits and through back navigation in the Monte Carlo record to the generated particle. Given the track density in the ATLAS detector it is possible that during the clusterisation process one PrepRawData object is constructed from several simulated hits that are caused by different generated particles. To account for this ambiguity, the PrepRawData truth collection is implemented as an associative container that allows this multiple relationship, even with given assignment probabilities (in C++ terms this is done using an STL multimap object).
- **Track Truth Association:** the task of associating the reconstructed track objects with the generated particles is defined defined by the IDetailedTrackTruthBuilder interface. A track can hereby correspond to one or many generated particles; since in full detector simulation many

hard interaction processes change the identifier of the particle, while for tracking the given chain of truth particles is in the optimal case still reconstructed as a whole, a one-to-many relationship has been established as default. The classification whether a track corresponds to a given truth trajectory is done using the `PrepRawData` truth collection created in the hit truth association process.

- `TrackParticle` **truth association:** the `TrackParticle` class is not in particular a component of NEWT, but builds the interface of track reconstruction to many analysis applications. Additionally it marks the representation of the track in the AOD containers[9]. Since in the ATLAS data model the `Track` is not contained anymore in the AOD, it is convenient to establish a dedicated truth association container for the `TrackParticle` objects, by simply forwarding the track truth information.

The Monte Carlo truth association is highly detector specific, therefore most of the `Algorithm` implementations are located in the sub-detector repositories. However, the common steering and the interfaces to be used by the different technologies are concentrated in the ATLAS Tracking realm.

Validation The validation of NEWT concentrates on two different topics: performance and reliability. Performance validation is in general done at various stages and is in many cases deeply woven into the sub-detector concepts. However, on a general track level it can be to some extent performed within the common tracking framework. A dedicated package, the TrkValidation concentrates therefore several `Algorithm` and `AlgTool` objects capable of filling histograms from tracking EDM objects from base class level, Tab. 2 presents an overview and a brief description of the components that can be found in the TrkValidaiton container.

Table 2: Container packages and brief description of their content in the TrkValidation package, packages dedicated for the validation of the vertexing performance are omitted since they are not particular to NEWT.

Container	Description
TrkValAlgs	steering algorithm for track based validation, including track parameter residuals and pulls, hit residuals and pulls (biased and unbiased), an `Algorithm` for track difference calculations, material validation
TrkValInteraces	interface definitions for `AlgTool` classes used within this context
TrkValTools	several `AlgTool` classes that are mainly used by modules from the TrkValAlgs package: a general residual and pull calculator, a hit position helper, etc.

Many of the performance actions rely on the Monte Carlo truth information and thus require the truth association to be done before. Hit residuals and pull distributions for hits on surfaces can however be calculated without truth association, since the *true* hit position is given by the measurement. The TrkValidation follows hereby the described structure of a common interface and dedicated implementations of `AlgTool` classes in the sub-detector realms to guarantee access to the underlying specific hit information. If the complete track information is provided, i.e. both the measurement and the track representation are given with covariance matrices, the unbiased residual can be calculated by applying an reverse update step to the track. Additional validation and statistics packages can be found in the sub-detector repositories such as e.g. the InDetRecStatistics package that focusses on a detailed validation of the pattern recognition efficiency and track reconstruction resolution in comparison with Monto Carlo truth information for the Inner Detector.

The reliability of the software is of similar importance as the tracking performance. In the ATLAS computing model, automatic test runs on standard input samples are performed for every new build of the software project to monitor sudden changes on the performance level. Additionally, the ATHENA framework provides several `Service` implementations to monitor memory consumption and leaks[10],

[9] In the ATLAS computing model, the event data exists on the persistent side in two levels, the Event Summary Data (ESD) and the highly compressed Analysis Object Data (AOD) that should be suitable for almost any analyses.

[10] The liability of memory leaks is one of the drawbacks of the data factory software design, since the latter requires a precise tracking of the object ownership.

timing performance and status code return values. The ATLAS New Tracking adds in addition an instance counting schema for the EDM object, performed by the `EventDataMonitor` (also located in the TrkValidation package) that can only be executed in a special debug mode.

4 The ATLAS New Tracking in the Inner Detector

In modern track reconstruction strategies there is no clear border between the classical modules *pattern finding* and *track fitting*. This is — on the one hand — due to the fact that many pattern finding strategies (contrary to a classical histogram based approach) nowadays incorporate a two stage pattern: a global pattern search, as well as a local pattern recognition where track fitting is already part of. On the other hand, many track fitters such as the combinatorial Kalman filter or the deterministic annealing filter incorporate an intrinsic pattern recognition during the fitting process. Thus, the full chain of pattern recognition and track fitting will be in the following described as a single unit.

The ID New Tracking currently covers two sequences, the main *inside-out* track reconstruction and a consecutive *outside-in* tracking. The primary pattern search concepts for both sequences have been to a large extent adopted from the already existing ATLAS ID reconstruction program `xKalman` [18], but integrated and accomplished by additional components in the common NEWT approach. A third sequence, the *second stage pattern* recognition for the finding of V0 vertices, kink objects due to bremsstrahlung and their associated tracks has been also deployed using the common tracking tools and EDM, but is not particular to the New Tracking approach. Section 4.1 and Sec. 4.2 will in the following describe the main concepts and tools used for establishing a full Inner Detector reconstruction chain under the New Tracking, while Sec. 4.3 highlights the special adoption of the ATLAS ID New Tracking for the third level trigger stage, the ATLAS *Event Filter* (EF). To distinguish the EF realisation from the standard ID reconstruction, latter will be in the following also referred to as *offline* reconstruction.

4.1 Inside-out Track Reconstruction

The primary ID pattern recognition follows an inside-out strategy for track finding. It is realised as a sequence of modules — each represented through a dedicated `Algorithm`— and described in more detail in the following sub-sections. Figure 2 shows an extended UML sequence diagram for the inside-out tracking. In a classical picture, many modules of the sequence can be divided into global pattern recognition and consecutive local pattern recognition that only works on the reduced output sample of the global search results.

4.1.1 `SpacePoint` Formation

The first step in the inside-out track reconstruction is the creation of three-dimensional representations of the silicon detector measurements, which are then called `SpacePoint` objects. For measurements with the pixel detector this is a very simple task, since the pixel modules provide a two-dimensional local measurement that is — using the constraint of the `Surface` representing the detector element – transformed into a `SpacePoint` by a simple *local-to-global* transformation. Single SCT clusters can not be transformed directly into a three-dimensional representation, since the precise measurement on an SCT module can only be given orthogonally to the silicon strip direction. The transformation to a three-dimensional point is therefore not constraint and a direct representation as a three-dimensional point can not be defined. However, the SCT detector is built with a sandwich module structure, i.e. two silicon modules are glued together back to back, but rotated by a stereo angle with respect to another, this relation — and together with a beam spot constraint — this can be used to construct the three-dimensional `SpacePoint`. The position of the beam spot is hereby automatically retrieved from the conditions data or can be set by hand (e.g. for cosmic ray reconstruction). In contrast to the pixel detector, where each cluster directly leads to a `SpacePoint` object, the SCT `SpacePoint` formation features an intrinsic noise suppression at the very first pattern stage, since it requires two different modules with separate readout for the creation of one single `SpacePoint` object.

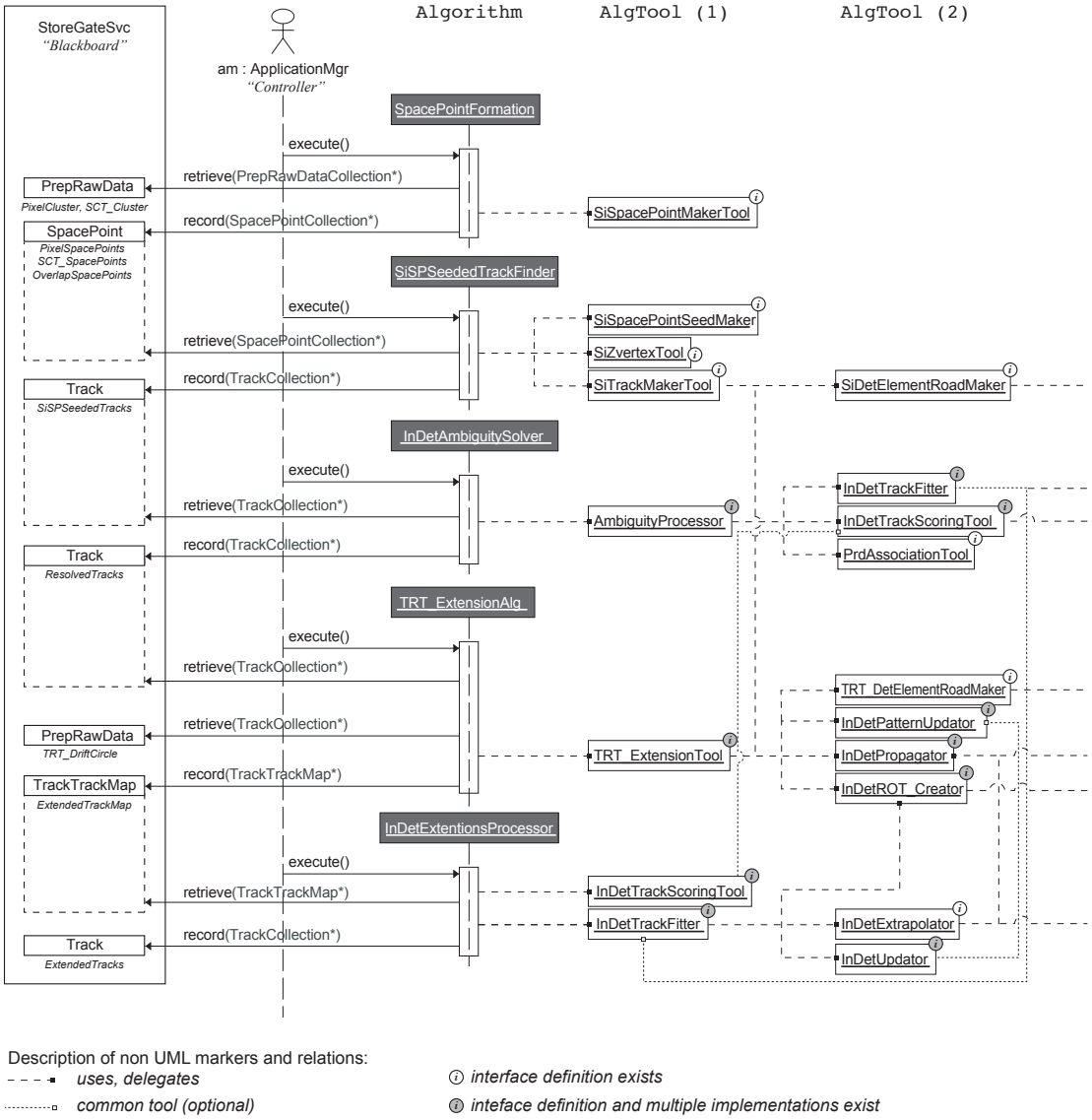


Figure 2: The main sequence of modules and some of the `AlgTool` classes used in the ATLAS New Tracking for the Inner Detector illustrated as an extended UML sequence diagram. Following a blackboard architecture structure, the transient event store (StoreGateSvc) acts as the *blackboard* for reading and writing of event data, the `ApplicationMgr` as the *controller* of the system. Only two levels of embedded `AlgTool` classes are shown, where dashed lines indicate a *uses* relationship, while the dotted line indicates possible sharing of `AlgTool` instances.

4.1.2 `SpacePoint` seeded Track Finding

The `SpacePoint` collections filled in the previous step are further processed for seeding the track candidate search in the Inner Detector. The `SiSPSeededTrackFinder Algorithm` represents this second module in the inside-out sequence. It can be divided into two different tasks, the track seed finding and the track candidate creation, based on the seeds found in the first step.

The seed search marks the global part of the pattern recognition and can be done in the following ways, using the `SiSpacePointSeedMaker`:

- **Seed search with z vertex constraint:** `SpacePoint` pairs from only the pixel detector are found in a first step and z vertices are built from these pairs using a dedicated `SiZVertexMaker`

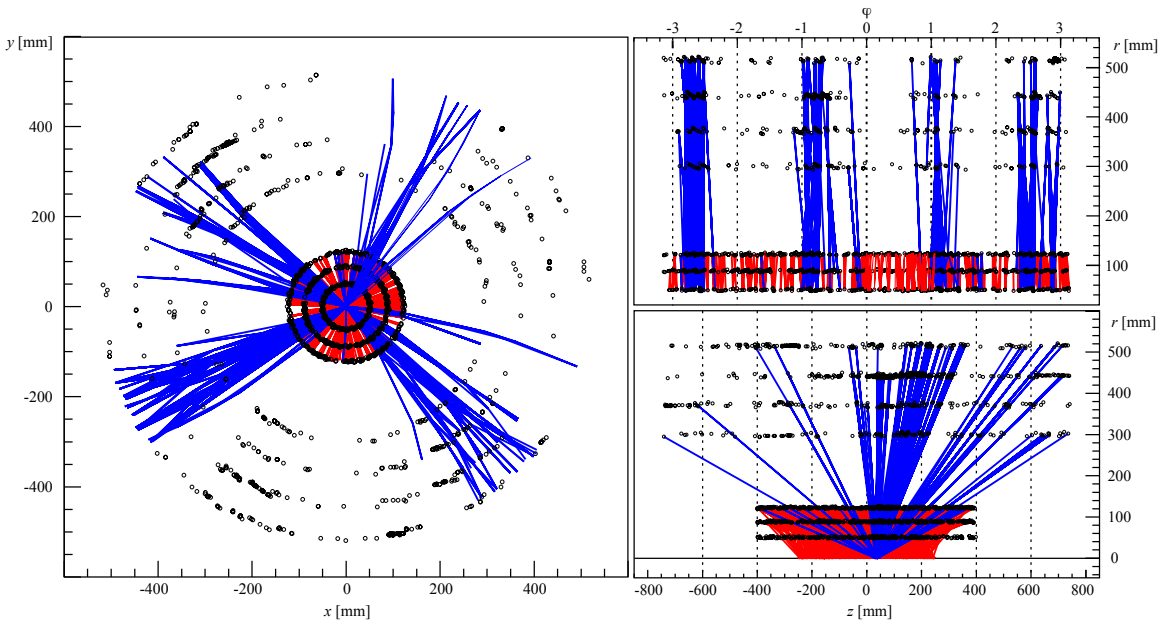


Figure 3: `SpacePoint` seeds consisting of two (short seeds) respectively three (long seeds) objects in the ATLAS Inner Detector barrel for a $t\bar{t}$ event, found with the z-vertex constraint seed search: the seeds consisting of two `SpacePoint` objects are used to determine z-coordinates of the predicted vertex positions. Only vertices within a defined range around the interaction point are used to constrain further seeds with three or more `SpacePoint` objects. For convenience, only seeds that are entirely in the barrel region are drawn.

`AlgTool`. The vertices are filled in histograms, keeping the seeds compatible with a given momentum and transverse impact range. A fast primary vertex search is performed and the primary vertex is used to further constrain the seeds with three or more space points. The tolerance region for predicted vertices from constructed seeds can hereby be chosen as a cut parameter. Figure 3 shows the seeds for vertex finding and track candidate search for an example $t\bar{t}$ event in the pixel and SCT barrel.

- **Unconstrained seed search:** The seed search can also be performed without the given z vertex constraint, which leads to a significantly higher number of initial track seeds (and in the following track candidates). The unconstrained seed search is evidently more time consuming, but more efficient to find tracks in events with loosely constraint primary vertices, such as $H \rightarrow \gamma\gamma$ decays or non-physical single track events with superimposed pile-up signatures. Figure 4 shows the z vertex distribution for an example $t\bar{t}$ event and Fig. 5 shows the resulting `SpacePoint` seeds found without z vertex constraint.

The z vertex scan is the standard `SpacePoint` seeded track search strategy in the ATLAS release 12.0.6, while for further production releases the unconstrained seeding is foreseen to be default in the ID NEWT track reconstruction.

Once the `SpacePoint` seeds are found, the road building process is started: the seeds provide already enough directional information to build roads of detector elements for the further search of associated hits to one track candidate. This marks the beginning of the local part of the silicon pattern recognition. At this stage, the `SpacePoint` objects are dissolved into the cluster objects of which they have been originally build from. This is, because the track candidate creation involves track fitting, which is in general performed on either `PrepRawData` or `RIO_OnTrack` level[11]. The cluster collections that contain also the clusters that have not been used to create `SpacePoint` objects are retrieved from the transient event store and those that are located on detector elements that build a road are used for the track candidate. A Kalman fitter-smoother formalism is used to simultaneously follow the trajectory

[11] The `SpacePoint` class, however, has been recently integrated into the `MeasurementBase` schema and could also be used for track fitting on this level. Since the creation of the `SpacePoint` objects include a projective error treatment, the fit on `RIO_OnTrack` level is more precise.

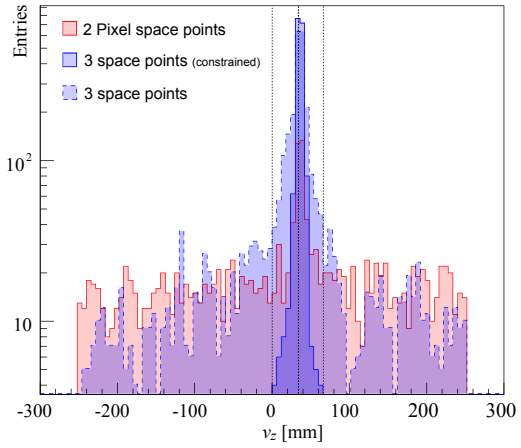


Figure 4: The distribution of the z vertex coordinate v_z of the projected vertices that originate from the different `SpacePointSeed` objects in the same event as shown in Fig. 3. The narrow distribution shows the v_z distribution for the seeds made of two pixel `SpacePoint` instances, where the other two show the same distributions for the seeds build from three `SpacePoint` objects for both, the z vertex constraint (solid) and the non-constraint (dashed) configuration. Additionally the region around the primary vertex found through the fast z vertex scan on the pixel seeds is shown.

and include successive hits in the track candidate fit. This approach is intrinsic to the Kalman filter formalism: it progressively updates the track information (including the covariances) and thus predicts precisely the track representation on the next measurement surface. Since, in general, a silicon detector element has more than one hit per event, the prediction leads to the most likely extension of the trajectory, while detecting outliers immediately via their large contribution to the χ^2 of the track.

`SpacePointSeed` objects do not necessarily lead to a track candidate, in contrary, only in about 10 percent of the cases the seed is successfully extended to a track candidate, stored in the common `Track` EDM format. The track finding from seeds, realised through the `SiSPSeededTrackMaker AlgTool` provides also the possibility to find more than one track candidate from a given seed, but this is a very rare case in the ATLAS ID event reconstruction.

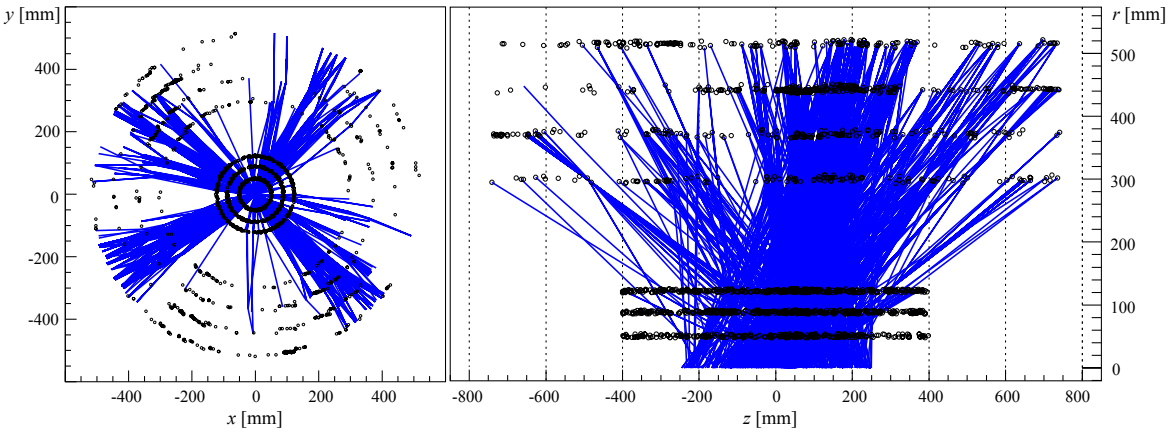


Figure 5: The same event as shown in Fig. 3 using the second silicon seed strategy without vertex constraint. The `SpacePoint` seed build of three objects are spread over a large z-range that leads to an increase of the track candidates for further processing.

4.1.3 Ambiguity Solving

The seeded track finding results in a very high number of track candidates, that have to be resolved before the extension into the outer TRT can be done. Many of these track candidates share hits, are incomplete or describe fake tracks, i.e. tracks where the majority of associated measurements do not originate from one single particle. The tracks have to be therefore ranked in their likelihood to describe the real trajectories of particles from the underlying physics event. A first step here is to refit

the track using the refined reconstruction geometry that has a detailed material description. However, the track fit only results in a global parameter, the χ^2 divided by the degrees of freedom, which is in most cases not appropriate to decide whether a track was a good or fake track. For the classification of tracks, a so-called *track scoring* strategy has been developed [19], that describes in addition to the fit quality morphologic parameters of the track: different characteristics of a track are hereby represented by a beneficial or penalty track score, which all together form an overall track score. In general, each hit associated with the track leads to a better score value to favor fully reconstructed tracks rather than small track segments. The measurements of different sub-detectors are, in general, weighted with different scores, preferring the precision measurements (e.g. pixel clusters) and downgrading measurements from less precise detector parts. Table 3 gives a qualitative overview of the different benefits and penalties of tracks found in the `SpacePoint` seeded track search, and Fig. 6 illustrates some of the track characteristics to be resolved in the SCT barrel detector.

Table 3: Track characteristics that lead to benefits or penalties in the ATLAS silicon detector track score.

Track characteristics	**Detector**	**Effect on the track score**
B layer hole	pixel	strong penalty
Layer hole	pixel	penalty
Overlap hit	pixel, SCT	strong benefit
Sensor hole	SCT	weak penalty
Layer hole (module)	SCT	strong penalty

Hits that are shared between tracks are — after the track scoring has happened — mainly assigned to the track with higher score, while the remaining track is being refitted without the formerly shared hit[12]. The refitted track is again scored and enters the remaining list of tracks to be evaluated. In an iterative procedure, the tracks with highest score are bundled and tracks that fall beyond a certain quality cut are neglected for further processing.

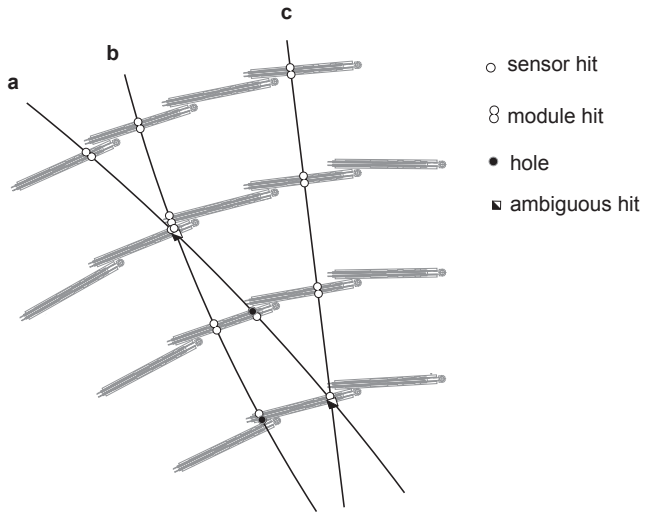


Figure 6: Simplified model of the ambiguity solving process, illustrated in the SCT Barrel. Tracks **a**, **b**, and **c** have been found through the seeded track finding, but share several hits. The χ^2/n_{dof} may not be appropriate to distinguish a true from a fake track, therefore dedicated track scoring that is optimised for each sub-detector is used. In the shown example, e.g. a *module hit* representing measurements on both sides of the SCT silicon detector are scored relatively higher than two single hits without associated backside module. Hits in a overlap region as for track **b** are in particular high scored, while holes on track, i.e. an expected hit that has not been found, lead to a penalty in the track score.

4.1.4 TRT Track Extension

The track (segment) extension from the silicon detector into the TRT consists of two modules, the `TRT_ExtensionAlg` and the `InDetExtensionProcessor`. The `TRT_ExtensionAlg Algorithm` steers the extension finding on a single track by track basis; the tracks found through the silicon seeds and

[12] In a recently introduced strategy, that has not yet been part of the ATLAS 12.0.6 release, hit sharing between tracks is allowed when the track fulfills dedicated quality criteria. This is to account for the fact that in the pixel system the readout creates an artificial ambiguity between hits that are joined together to one readout element (ganged pixels).

resolved by the ambiguity processing are used as an input to find compatible sets of TRT measurements that are further processed as candidate extension. Again, the `TRT_ExtensionAlg` simply delegates this task to a dedicated `AlgTool`, represented through the `ITRT_TrackExtensionTool` interface. The silicon-only track must hereby not be modified, the association of the TRT hits are therefore purely extensions and is not done by a combination. Figure 7 shows the extensions of the silicon seeded tracks into the TRT detector for a sample $t\bar{t}$ event.

Two concrete implementations of the track extension tool exist:

- `TRT_ExtensionTool_xk`: the `TRT_ExtensionTool_xk` is the standard implementation used in the Inner Detector New Tracking. It follows a classical approach starting with road finding through track extrapolation, and — using the hit coordinates expressed in $r-\phi$ in the barrel, and $r-z$ in the endcap region, respectively — performs a line fit to estimate whether the hit is compatible with the silicon track seed or not.

- `TRT_ExtensionTool_DAF`: this extension tool is designed using the deterministic annealing filter (see Sec. 3.3.3) that is optimised for very high hit densities. The DAF strategy also starts with the road building (and uses hereby the same `AlgTool` as the first extension strategy), but follows a different philosophy for the hit finding and hit assignment; TRT measurements within the road are grouped together on the same readout element (or on planes perpendicular to the extrapolated track, depending of the initial configuration) and represented as one input object to the track fit. The different hits within the group are weighted by their likeliness to represent the true hit, while the weights correspond mainly to the distance of the hit from the trajectory prediction (i.e. the residual).

The `TRT_ExtensionAlg` produces a map of the seeded silicon tracks and the found extensions, if no extension is found through the concrete version of the `ITRT_TrackExtensionTool` the second map entry is simply left empty. This map is written to the transient event store and successively picked up by the second `Algorithm` in the TRT extension module, the `InDetExtensionProcessor`. This `Algorithm` is responsible for evaluating the extended track with respect to the pure silicon track. The comparisons of the two tracks is based on a combined track refit and then done using the track scoring mechanism, comparing the track score of the original track with the one after refitting. Unlike the track extension `Algorithm` which is not allowed to change the initial silicon track, the refit in the `InDetExtensionProcessor` can modify the silicon hits by flagging them as outlier measurements. In case that the track score of the silicon track is higher than the extended version, the silicon track is kept and the TRT hits are put as outlier measurements onto this track. The `InDetExtensionProcessor` can be configured to work optionally with the DAF specific extension algorithm. In this configuration the DAF is specified as the track fitter used for the evaluation of the track extension, applying the annealing schema on the multiple measurements associated to the track in the `TRT_ExtensionTool_DAF`.

4.2 Outside-in Track Reconstruction

The inside-out sequence of the ID New Tracking relies on a track seed found in the silicon detector. In the track reconstruction process, some of these initial track seeds may not be found or do even not exist: ambiguous hits can shadow the track seed in the silicon and prevent the score of the silicon seeded track to survive the ambiguity processor on the one hand, and on the other hand, tracks coming from secondary decay vertices further inside the Inner Detector volume (e.g. K_s decays) or from photon conversions may not have any or only insufficient silicon hits to comply with the inside-out sequence. In a third pattern, substantial energy loss — mostly of electrons — at outer radii of the silicon `SpacePoint` seeded track and not known to the road building may guide the `TRT_TrackExtensionTool` into a *wrong* direction, such that no corresponding TRT hits are found. The ID New Tracking realisation will establish therefore a second sequence in track reconstruction, following an outside-in approach. The sequence will be realised as two different modules, starting with a dedicated segment finding algorithm and a successive back tracking of the segments into the silicon detector. In release 12.0.6 only the first part has been included and is discussed below.

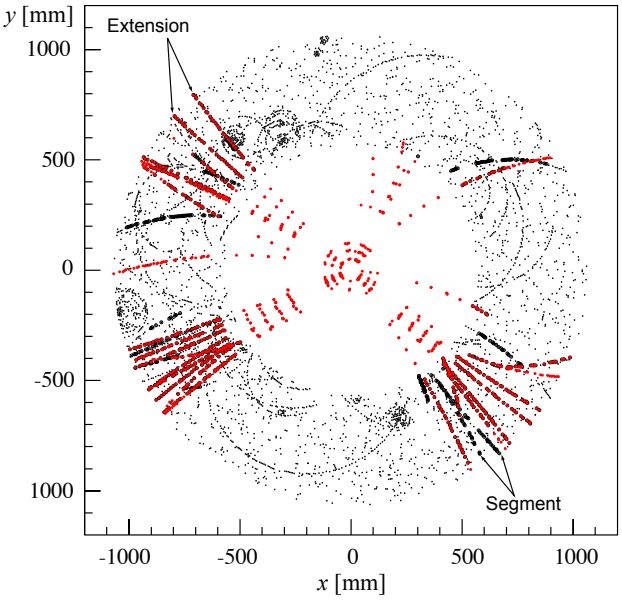


Figure 7: The same $t\bar{t}$ event as shown in Fig. 3 for the two possible TRT hit associations (only the barrel measurements): the brighter colored hits show the extensions that originate from following the `SpacePoint` seeded tracks into the TRT, the silicon `SpacePoint` objects are also shown in the same color. The black circles mark hits that have been associated to TRT segments, which builds the start point of the back tracking application. The particular power of the back tracking approach is to find the additional track segments, that have not been found through the inside-out sequence, simply for the fact that no appropriate silicon seed did exist for the further extension process. This is mainly due to strong energy loss of the particle, or due to the fact that the track segments originate from photon conversions or other decay vertices inside the Inner Detector volume.

4.2.1 TRT Segment Finding

The currently existing TRT segment finder implementation in the ID New Tracking realm is based on the outside-in track reconstruction strategy taken from the legacy `xKalman` program. It follows a two step procedure, starting with a global pattern search and a subsequent local pattern recognition with intrinsic track segment building. Since the TRT drift tube measurements do not provide any information about the coordinate along the straw direction, `SpacePoint` objects can not be built and the global pattern recognition has to be done in projective planes. Evidently, the most adequate projection planes for the TRT geometry have been chosen: the $r-\phi$ plane in the TRT barrel region and the $r-z$ plane in the TRT endcap part, where the single straws fan out on disc structures. Assuming that the tracks originate roughly from the primary interaction region, track segments from tracks with transverse momentum greater than 500 MeV appear as almost straight lines in the $r-\phi$ and rigorous straight lines in the $z-\phi$ projection. There exist many techniques to find straight line patterns. A very common one in high energy physics event reconstruction, the Hough transform [20], is used to find the hit pattern: it is based on the simple fact that by transforming the projection plane $r-\phi$ (or $z-\phi$, respectively) into the parameter space of the straight line — in this specific case identified as the initial azimuthal angle ϕ_0 and the inverse momentum parameter c_T (respectively c_z) — the points associated with one line are transformed into one single cell, since they satisfy the same line parameterisation. The global track segment search thus can be reduced to the local maximum finding in a two-dimensional histogram. To reduce the number of overlaying track segments, this histograming process is done for several η slices of the TRT detector. The missing hit information along the drift tubes in the TRT, however, results in the fact that hits have to be in general considered in several different slices. This relation has to be tracked and resolved by a simple maximisation of the the straw hits per found track candidate. Figure 8 shows a two-dimensional histogram for an example η slice in the Hough space.

Local Pattern Recognition and Event Sample Cleaning The histogram method provides a set of track segment candidates that are further processed in a second step of the TRT segment finding. Whereas the global hough transform uses the straw center position for the finding of compatible sets of hits, the drift time information is also used in the local pattern recognition process: using a Kalman filter-smoothing formalism the track segments are build and the final collection of `TrackSegment` objects are written to the transient event store.

In many cases, the `TrackSegment` finding will pick up segments that have been already successfully associated to tracks found in the silicon detector by the extension `Algorithm`. To save CPU time the segment finding is planned to work on a cleaned out hit sample. For the event cleaning, an association

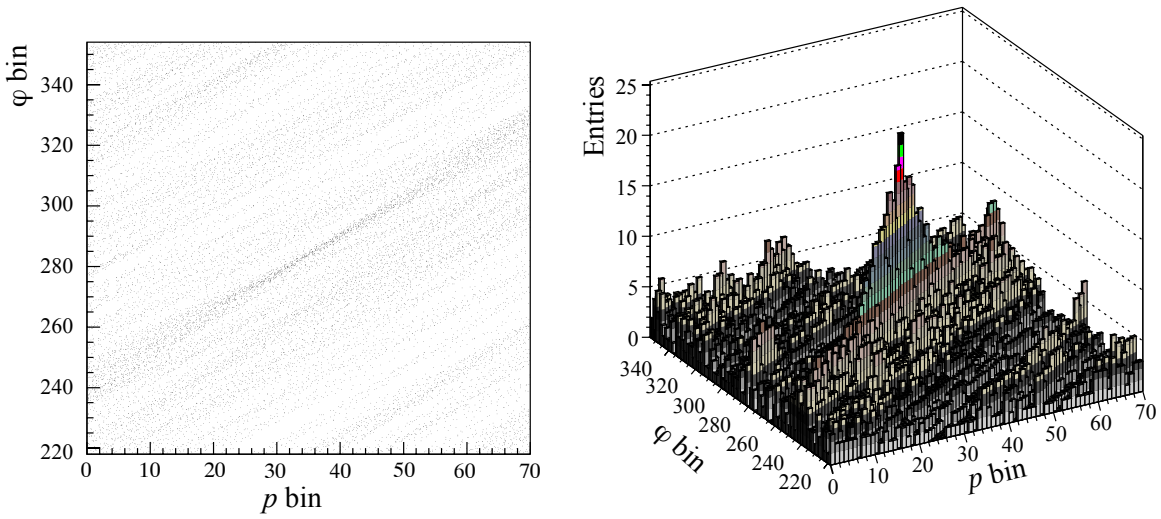


Figure 8: Straw representations in a part of the parameter space (ϕ bins and momentum bins) after applying the Hough transform. The Illustration shows a subset of the full parameter space that is divided in the standard configuration into 500 bins in the azimuthal direction, while trying 70 momentum hypotheses. The left side shows the Hough space in a scatter plot, while the right histogram shows the same subset in a three-dimensional version, where the most probably (ϕ_0, c_T) hypothesis can be clearly identified; the plot is restricted to a single η slice.

tool — the `PRD_AssociationTool` — that is also used in the ambiguity processing phase can be taken to find the hit-track relation.

The second step of the outside-in approach, the backtracking of the TRT `TrackSegment` objects into the silicon Detector is planned to be integrated with release 13.0.0 and will be presented in a separate document [21].

4.3 The Event Filter Realisation

One special aim that has been followed during the development of the ATLAS ID New Tracking software was to be able to provide the same track reconstruction chain to the ATLAS High Level Trigger realm. This is achieved through the modular NEWT design which allows to replace time-critical components and full-featured offline modules by trigger-specific implementations.

The ATLAS trigger system consists of a three step triggering system, with one pure hardware-based LVL1 trigger, followed by the software LVL2 and the Event Filter (EF) [22]. The trigger strategy starts from a very fast calorimeter and Muon Spectrometer based region of interest (ROI) estimation. The ROIs found in the LVL1 might be further refined in the LVL2 trigger, where the calorimeter clustering is repeated with higher granularity and first tracking stand-alone algorithms are run in the ID and the Muon System. The Event Filter is the last step in the ATLAS Trigger chain. It is a pure software trigger, working on the output of the LVL2 trigger objects. These objects already incorporate a hypothesis of the event morphology and steer the EF to run in one of several pre-defined slices, among which are the *electron*, *muon*, *b-jet*, *b-physics*, *tau* and *gamma* slices. Each of these slices contains a very similar `Algorithm` sequence as the ID inside-out tracking and are followed, depending on the given slice by dedicated event reconstruction algorithms including vertex finding, b-tagging or electron processing. In the EF realisation of NEWT dedicated `Algorithm` classes steer the underlying `AlgTool` objects with ROI seeded input collections. The used `AlgTool` classes are directly taken from the offline reconstruction chain but operated in a ROI seeded mode, where the trigger slice defines the width of the ROI. Given the time constraint of the EF, which is of about 1 second of total process time per event, the NEWT `AlgTool` classes are configured using simplifications such as the default z vertex scan in the `SpacePoint` seed finding or the use of the fast vertex fitter in the post processing time. For each trigger slice, a dedicated output collection is written to the transient event store, while overlapping can take place in both ROI regimes and multiple appearance in the different pre-defined

slices. Figure 9 shows an example simulated $t\bar{t}$ event reconstructed with the offline Inner Detector New Tracking, and the ROI seeded event filter reconstruction, illustrated with the ATLAS event display ATLANTIS [23].

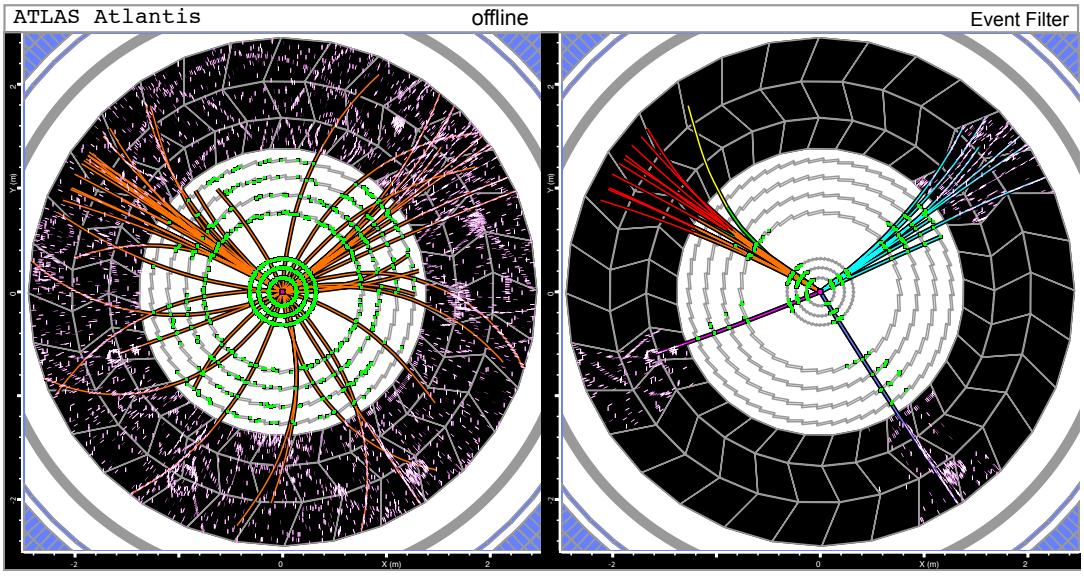


Figure 9: An simulated $t\bar{t}$ event reconstructed with the offline and Event Filter versions of the ATLAS New Tracking. The Event filter has been executed here in the electron slice which determines a dedicated region of interest size in which the full NEWT inside-out sequence is performed. The four regions of interest found by the prior LVL1 and LVL2 trigger and associated to an electron hypothesis can be clearly distinguished.

4.4 NEWT in the Combined Test Beam and Commissioning Setup

A complete independent New Tracking sequence, the CTBTracking [24], has been initially developed for the combined test beam run in 2004 (CTB2004). The CTB2004 was the first time that the ATLAS offline reconstruction software had to reconstruct real data of a complete slice of the ATLAS detector. The CTBTracking was the first complete reconstruction chain that was realised through the modular NEWT design and served as a prototype for the new software model. It incorporates a combinatorial track finding based on silicon space point samples that is valid for a small number of track candidates and is based on the `GlobalChi2Fitter`, see Sec. 3.3.3. Intermediate fits of the track candidates are used to reduce the number of track candidates and a χ^2 based ambiguity solving mechanism disentangles track candidates with ambiguous hit patterns. The CTBTracking has been extensively used for the track reconstruction and alignment studies of the CTB2004 and has been extended to be applicable for the ongoing ATLAS commissioning runs with cosmic ray tracks. Figure 10 shows a cosmic muon track reconstructed with the CTBTracking.

5 NEWT for Combined Reconstruction

Initially deployed in the ATLAS Inner Detector, common components and modules of the New Tracking have already spread widely into software applications for the Muon System and combined reconstruction. The common ATLAS tracking EDM has also become the input event data format of all track reconstruction algorithms of the Muon System and the ATLAS `TrackingGeometry` has been expanded to a full description of the ATLAS detector. Although no complete reconstruction chain based on the New Tracking concepts exists so far in the Muon System or combined reconstruction, the following section will cover some of the main tools used in combined reconstruction.

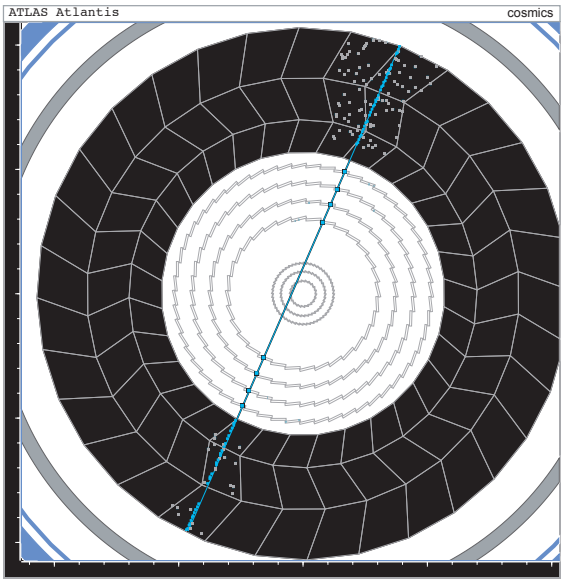


Figure 10: A real cosmic ray event found by the CTBTracking package and shown with the ATLAS event display ATLANTIS. The CTBTracking has been initially developed for the combined test beam run and was the first reconstruction sequence entirely build from New Tracking components using the common event data model. The CTBTracking has been also adopted to serve the needs for reconstructing cosmic ray tracks.

5.1 Calorimeter Impact and Vertex Expression

Based on the new ATLAS extrapolation engine, the `TrackToCalo` application was the first in a list of following combined muon reconstruction tools[13] that have been based on the tracking EDM and common tracking tools. The `TrackToCalo AlgTool` is realised as a wrapper of the `Extrapolator AlgTool` and prepares the input data for the track-to-cluster comparison: a specified track collection is retrieved from the transient event store, for each of the tracks that are contained by the given collection an extrapolation is performed to the different sampling layers of the calorimeter and the cells with energy deposit are collected in a given isolation cone. The different coordinate definitions between the Inner Detector and the calorimeter have to be hereby taken into account, which is performed in the Reconstruction repository and completely shielded from the common Tracking tools. `TrackToCalo` is realised as a central `Algorithm` that performs the comparison per default for every track. However, many applications will be only interested in a particular subset of the track collection. For this purpose, the single track operation is encapsulated in a dedicated `AlgTool`, the so-called `ExtrapolTrackToCaloTool` which can be used independently by any client.

A very similar `AlgTool`, the `TrackToVertex`, performs the extrapolation of a track to a given vertex position. It simply wraps the extrapolation engine preparing a user-friendly interface.

5.2 Combined Muon Refitting

The combined fit of a track that originates from a muon with the use of both independent tracking devices is an outstanding goal of the new ATLAS track reconstruction. Clearly, two main strategies can be identified to combine tracks from the Inner Detector with those found in the Muon System: a single combination of the two tracks that are represented with respect to the same reference surface (e.g. both track segments are in a perigee representation) based on a χ^2 minimisation, or the complete dissolving of the track into the hit objects and performing a combined track fit. While the first robust method is widely used in ATLAS and achieves a satisfactory performance, the latter technique has not been fully established yet using NEWT in the ATLAS offline reconstruction. A combined refit is not in particular expected to gain on the track resolution substantially when being compared to resolution given by the respectively better single sub-detector[14], but should help to recover tails in the combined track resolution, since the combined fit allows to redefine outlier measurements or to re-weight hit contributions to the final track representation. The complete track refit will in addition play an important role in the global alignment of the ATLAS detector. Various inevitable requirements for the establishment of combined refitting have been met already: the common `Track` class as a well

[13] Most of these tools are located in the RecoTools CVS package of the ATLAS offline repository.

[14] It can be shown that the ID performance in terms of momentum resolution is superior to the MS for low momenta, while for high momenta the outer Muon Spectrometer outplays the Inner Detector due to the higher magnetic field and larger dimension.

defined output of track reconstruction programs (including the entire common hit and measurement descriptions), the creation and deployment of a global reconstruction geometry and the realisation of tracking tools that work on complete technology independent base class level. First attempts of combined refitting have been done [25] and show promising results and the validity of the applied software design.

6 Conclusion and Outlook

Four years after the public release of the *Final Report of the Reconstruction Task Force*, a new modular track reconstruction software has been established that includes a common EDM, an underlying reconstruction geometry and is based on well-defined interfaces in a component pattern design. A first complete model of the inside-out track reconstruction has been deployed in the Inner Detector and is competitive in terms of performance and CPU time consumption to prior ATLAS reconstruction programs, while providing an open and interactive model for future modifications and adoptions. The second sequence based on common tracking tools, the outside-in tracking, is close to completion. NEWT builds in addition the backbone of the ATLAS Inner Detector Event Filter by providing the seeded pattern recognition and fitting tools used in the Event Filter trigger slices.

Moreover, all track reconstruction algorithms now provide the EDM objects as the output data, allowing common validation and analysis applications to work independently on the reconstruction chain used for track finding and fitting.

6.1 Outlook

The full migration of the previous track reconstruction programs into the New Tracking schema is an ongoing effort of the ATLAS offline software project. NEWT was designed to allow both, the easy introduction of newly developed concepts as modules to the common tracking effort and the re-integration of the existing well-performing algorithms from past reconstruction packages. Latter is deliberately foreseen without wiping the identity (in both concepts and performance) of these well-tested algorithms. Recently, main components such as the intersection algorithms from the Inner Detector stand-alone `iPatRec` [26] application have been integrated into the extrapolation engine of New Tracking, and is followed by an ongoing effort to employ internally the common tracking EDM. Finally, the track fitting `AlgTool` from `iPatRec` will be integrated as another implementation of the common `ITrackFitter` interface to provide full flexibility to the user of the ATLAS offline reconstruction.

A second big field of further development is the realisation of combined reconstruction and recovery and detection strategies of bremsstrahlung radiation using the ATLAS New Tracking tools. Clearly many parts of NEWT that have been described in this document can become parts of new `Algorithm` chains that concentrate on higher stage pattern recognition, particle identification and event topology classifications.

Acknowledgments

The authors would thank the various contributers to both fields the conceptual design as well as the software implementation of the ATLAS New Tracking.

A Appendix

A.1 Typesetting and Conventions

The following type setting conventions are followed throughout this document: software packages within the ATLAS offline software repository [27] are written in Sans-Serif face, C++ or python class names are written in `Courier` face. Namespace definitions as used in the software repository are omitted in this document for readability. Table 4 gives an exhaustive list of the active container packages in the Tracking repository, adding short descriptions of their content.

The software illustrations comply, in general, with Unified Modeling Language (UML) [28] standards, extensions to the UML standard are identified explicitly in the figure capture.

Table 4: Active container packages and brief description of their content in the Tracking CVS repository

Container	description
TrkAlgorithms	Tracking specific `Algorithm` classes to be executed once per event, i.e. truth association and conversion to persistent objects
TrkCBNT	legacy package to fill the Combined Ntuple (CBNT)
TrkDetDescr	container package for the reconstruction geometry, concentrating the installed classes and geometry builders
TrkDoc	documentation package
TrkEvent	container package for all tracking EDM base classes, concrete implementations may be found in the InnerDetector or MuonSpectrometer, respectively
TrkEventCnv	converter packages for persitification process
TrkExtrapolation	container package for the extrapolation engine
TrkFitter	track fitters implementing the `ITrackFitter` interface
TrkMagneticField	magnetic field access tools, magnetic field parameterisations
TrkTools	common tracking tools other than fitting, extrapolation, vertexing
TrkUtilityPackages	mathematical utilities, such as a Hough transform
TrkValidation	common validation package define on EDM base class level
TrkVertexFitter	vertex fitting interface and implementations

A.2 Index of Abbreviation and Acronyms

AKF *Alignment Kalman Filter* is a dedicated extension of the Kalman Filter that incorporates geometrical updates to the reconstruction geometry

AOD *Analysis Object Data* is the compressed event data dedicated for physics analysis

ATLAS *A Toroidal LHC ApparatuS*

CSC *Cathode Strip Chambers* are a technology used in the Muon System

CVS *Concurrent Versions System* is the used code archive and versioning software in ATLAS

CTB2004 *Combined Test Beam* was a combined test of a full sectorial ATLAS slice in 2004

DAF *Deterministic Annealing Filter* is an extension of the Kalman filter with an additional annealing schema

DKF *Distributed Kalman Filter* is a fast version of the Kalman filter designed for the Trigger and Event Filter

DNA *Dynamic Noise Adjustment* is a special extension of the Kalman filter in ATLAS

EDM *Event Data Model* - nomen est omen

EF *Event Filter* is the third level, software based trigger in ATLAS

ESD *Event Summary Data* is the uncompressed event information

GSF *Gaussian Sum Filter* is a special multi-Gaussian extension of the standard Kalman Filter

ID *Inner Detector* is the inner tracking device of the ATLAS detector

LVL1 *First Level Trigger* in ATLAS, purely hardware based

LVL2 *Second Level Trigger* in ATLAS

MS *Muon System* is the outer tracking device for muons of the ATLAS detector

MDT *Monitored Drift Tubes* are used in the ATLAS Muon System

NEWT *New Tracking*

RCP *Resistive Plate Chambers* are another technology in the ATLAS Muon System

ROI *Region of Interest* a defined region for further processing in the Trigger chain

RTF *Reconstruction Task Force* - a task force held in 2003 focussing on the evolution of the ATLAS event reconstruction

SCT *Semi Conductor Tracker* is the middle part of the ATLAS Inner Detector

STL *Standard Template Library* is a widely used container library for C++

TGC *Thin Gap Chambers* are used in the ATLAS Muon System

TRT *Transition Radiation Tracker* is the outermost part of the ATLAS Inner Detector

UML *Unified Modeling Language* is a standard in software modeling and illustration

References

[1] V. Boisvert et al, *Final Report of the ATLAS Reconstruction Task Force*, ATLAS Note, ATL-SOFT-2003-010, 2003.

[2] F. Akesson et al, *The ATLAS Tracking Event Data Model*, ATLAS Public Note, ATL-SOFT-PUB-2006-004, 2006.

[3] ATLAS Quality Assurance Group, *Atlas C++ Coding Standard Specifications*, ATLAS Note, ATL-SOFT-2002-001, 2002.

[4] Atlas Collaboration, *ATLAS Computing Technical Design Report*, ATLAS TDR, CERN-LHCC-2005-022, 2005.

[5] G. Barrand et al., *GAUDI - A software architecture and framework for building LHCb data processing applications*, Proc. of CHEP 2000, 2000.

[6] E. Obreshkov , et. al, *Organization and management of ATLAS software releases*, ATLAS Public Note, ATL-SOFT-PUB-2006-008, 2006.

[7] Lixin Tao, Xiang Fu and Kai Qian, *Software Architecture Design - Methodology and Styles*, Stipes Publishing L.L.C., ISBN 1-58874-621-6, 2006.

[8] A. Salzburger, M. Wolter and S. Todorova, *The ATLAS Tracking Geometry Description*, ATLAS Public Note, ATL-SOFT-PUB-2007-004, 2007.

[9] A. Salzburger, *The ATLAS Extrapolation package*, ATLAS Public Note, ATL-SOFT-PUB-2007-005, 2007.

[10] R. Frühwirth et al., *Application of Kalman Filtering to Track and Vertex Fitting*, Nucl. Inst. Meth., A 262, 1987.

[11] V. Kartvelishvili, *Electron bremsstrahlung recovery in ATLAS*, Proceedings of the 10th Topical Seminar on Innovative Particle and Radiation Detectors (IPRD06), Siena, 2006.

[12] R. Frühwirth, A. Strandlie, T. Todorov and M. Winkler, *Recent results on adaptive track and multitrack fitting*, Nucl. Inst. Meth., Volume 502, 2003.

[13] S. Fleischmann, *Track Reconstruction in the ATLAS Experiment : The Deterministic Annealing Filter* , CERN-THESIS-2007-011, 2007.

[14] R. Frühwirth, A. Strandlie, *Track finding and fitting with the Gaussian-sum Filter*, Proc. of CHEP 1998, 1998.

[15] R. Frühwirth, T. Todorov and M. Winkler, *Estimation of detector alignment parameters using the Kalman fitter with annealing*, Journal of Physics G: Nuclear and Particle Physics, 29, 2003.

[16] L. Buggem J. Myrheim, *Tracking and Track Fitting*, Nucl. Inst. Meth., 179, 1981.

[17] N. Konstantinidis et al., *Fast Tracking for the ATLAS LVL2 Trigger*, Proceedings of CHEP2004, Interlaken, 2004.

[18] I. Gavrilenko, *Description of Global Pattern Recognition Program (XKALMAN)*, ATLAS Internal Note, ATL-INDET-97-165, 1997.

[19] D. Wicke, *A New Algorithm For Solving Track Ambiguities*, DELPHI 98-163, PROG 236 TRACK 92, 1998.

[20] R. Duda and P. Hart, *Use of the Hough Transformation to Detect Lines and Curves in Pictures*, Comm. ACM, Vol. 15, 1972.

[21] T. Koffas, *private communication.*

[22] The ATLAS Collaboration, *ATLAS High Level Trigger, Data Acquisition & Controls - Technical Design Report*, CERN-LHCC-2003-021, 2003.

[23] ATLANTIS homepage, *http://cern.ch/atlantis*

[24] T. Cornelissen, *CTBTracking: track reconstruction for the testbeam and cosmics*, ATLAS Internal Note, ATL-INDET-INT-2006-001, 2006.

[25] T. Cornelissen, *Track Fitting in the ATLAS Experiment*, PhD Thesis, ISBN 90-6464-051-3, 2006.

[26] R. Clifft, A. Poppleton, *IPATREC: inner detector pattern-recognition and track-fitting*, ATLAS Internal Note, Soft-94-009, 1994.

[27] Atlas offline software repository, *http://atlas-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/offline/*

[28] M. Fowler, K. Scott, *UML Distilled*, Addison Wesley Longman, ISBN 0-201-32563-2, 1997.